CHƯƠNG IV - LUỒNG

IV.1 Mục đích

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Các khái niệm gán với hệ điều hành đa luồng
- Các vấn đề liên quan với lập trình đa luồng
- Các ảnh hưởng của luồng tới việc thiết kế hệ điều hành
- Cách thức các hệ điều hành hiện đại hỗ trợ luồng

IV.2 Giới thiệu

Mô hình thực thi trong chương 3 giả sử rằng một quá trình là một chương trình đang thực thi với một luồng điều khiển. Nhiều hệ điều hành hiện đại hiện nay cung cấp các đặc điểm cho một quá trình chứa nhiều luồng (thread) điều khiển. Trong chương này giới thiệu các khái niệm liên quan với các hệ thống máy tính đa luồng, gồm thảo luận Pthread API và luồng Java.

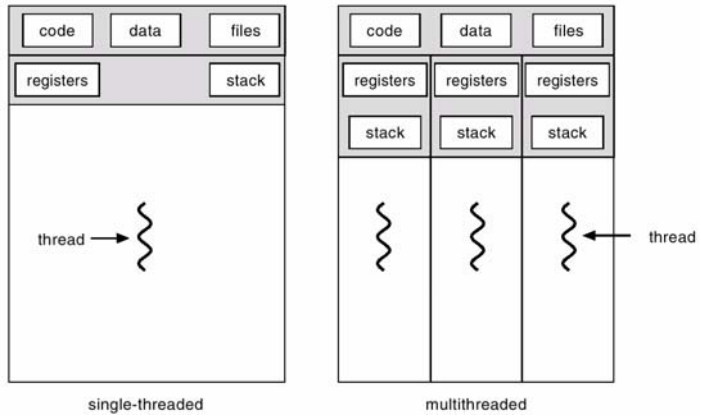
Chúng ta sẽ xem xét nhiều vấn đề có liên quan tới lập trình đa luồng và nó ảnh hưởng như thế nào đến thiết kế của hệ điều hành. Cuối cùng, chúng ta sẽ khám phá nhiều hệ điều hành hiện đại hỗ trợ luồng tại cấp độ nhân như thế nào.

IV.3 Tổng quan

Một luồng thường được gọi là **quá trình nhẹ** (lightweight proces-LWP), là một đơn vị cơ bản của việc sử dụng CPU; nó hình thành gồm: một định danh luồng (thread ID), một bộ đếm chương trình, tập thanh ghi và ngăn xếp. Nó chia sẻ với các luồng khác thuộc cùng một quá trình phần mã, phần dữ liệu, và tài nguyên hệ điều hành như các tập tin đang mở và các tín hiệu. Một quá trình truyền thống (hay quá trình nặng) có một luồng điều khiển đơn. Nếu quá trình có nhiều luồng điều khiển, nó có thể thực hiện nhiều hơn một tác vụ tại một thời điểm. Hình VI.1 hiển thị sự khác nhau giữa quá trình đơn luồng và quá trình đa luồng.

IV.3.1 Sự cơ động

Nhiều gói phần mềm chạy trên các máy để bàn PC là đa luồng. Điển hình, một ứng dụng được cài đặt như một quá trình riêng rẻ với nhiều luồng điều khiển. Một trình duyệt Web có thể có một luồng hiển thị hình ảnh, văn bản trong khi một luồng khác lấy dữ liệu từ mạng. Một trình soạn thảo văn bản có thể có một luồng hiển thị đồ họa, luồng thứ hai đọc sự bấm phím trên bàn phím từ người dùng, một luồng thứ ba thực hiện việc kiểm tra chính tả và từ vựng chạy trong chế độ nền.



Hình 0-1 Quá trình đơn và đa luồng

Trong những trường hợp cụ thể một ứng dụng đơn có thể được yêu cầu thực hiện nhiều tác vụ đơn. Thí dụ, một trình phục vụ web chấp nhận các yêu cầu khách hàng như trang web, hình ảnh, âm thanh, ..Một trình phục vụ web có thể có nhiều (hàng trăm) khách hàng truy xuất đồng thời nó. Nếu trình phục vụ web chạy như một quá trình đơn luồng truyền thống thì nó sẽ có thể chỉ phục vụ một khách hàng tại cùng thời điểm. Lượng thời gian mà khách hàng phải chờ yêu cầu của nó được phục vụ là rất lớn.

Một giải pháp là có một trình phục vụ chạy như một quá trình đơn chấp nhận các yêu cầu. Khi trình phục vụ nhận một yêu cầu, nó sẽ tạo một quá trình riêng để phục vụ yêu cầu đó. Thật vậy, phương pháp tạo ra quá trình này là cách sử dụng thông thường trước khi luồng trở nên phổ biến. Tạo ra quá trình có ảnh hưởng rất lớn như được trình bày ở chương trước. Nếu quá trình mới sẽ thực hiện cùng tác vụ như quá trình đã có thì tại sao lại gánh chịu tất cả chi phí đó? Thường sẽ hiệu quả hơn cho một quá trình chứa nhiều luồng phục vụ cùng một mục đích. Tiếp cận này sẽ đa luồng quá trình trình phục vụ web. Trình phục vụ sẽ tạo một luồng riêng lắng nghe các yêu cầu người dùng; khi yêu cầu được thực hiện nó không tạo ra quá trình khác mà sẽ tạo một luồng khác phục vụ yêu cầu.

Luồng cũng đóng một vai trò quan trọng trong hệ thống lời gọi thủ tục xa (remote process call-RPC). Như đã trình bày ở chương trước, RPCs cho phép giao tiếp liên quá trình bằng cách cung cấp cơ chế giao tiếp tương tự như các lời gọi hàm hay thủ tục thông thường. Điển hình, các trình phục vụ RPCs là đa luồng. Khi một trình phục vụ nhận một thông điệp, nó phục vụ thông điệp dùng một luồng riêng. Điều này cho phép phục vụ nhiều yêu cầu đồng hành.

IV.3.2 Thuận lợi

Những thuận lợi của lập trình đa luồng có thể được chia làm bốn loại:

- **Sự đáp ứng**: đa luồng một ứng dụng giao tiếp cho phép một chương trình tiếp tục chạy thậm chí nếu một phần của nó bị khóa hay đang thực hiện một thao tác dài, do đó gia tăng sự đáp ứng đối với người dùng. Thí dụ, một trình duyệt web vẫn có thể đáp ứng người dùng bằng một luồng trong khi một ảnh đang được nạp bằng một luồng khác.
- **Chia sẻ tài nguyên**: Mặc định, các luồng chia sẻ bộ nhớ và các tài nguyên của các quá trình mà chúng thuộc về. Thuận lợi của việc chia sẽ mã là nó cho phép

một ứng dụng có nhiều hoạt động của các luồng khác nhau nằm trong cùng không gian địa chỉ.

- **Kinh tế**: cấp phát bộ nhớ và các tài nguyên cho việc tạo các quá trình là rất đắt. Vì các luồng chia sẻ tài nguyên của quá trình mà chúng thuộc về nên nó kinh tế hơn để tạo và chuyển ngữ cảnh giữa các luồng. Khó để đánh giá theo kinh nghiệm sự khác biệt chi phí cho việc tạo và duy trì một quá trình hơn một luồng, nhưng thường nó sẽ mất nhiều thời gian để tạo và quản lý một quá trình hơn một luồng. Trong Solaris 2, tạo một quá trình chậm hơn khoảng 30 lần tạo một luồng và chuyển đổi ngữ cảnh chậm hơn 5 lần.
- **Sử dụng kiến trúc đa xử lý**: các lợi điểm của đa luồng có thể phát huy trong kiến trúc đa xử lý, ở đó mỗi luồng thực thi song song trên một bộ xử lý khác nhau. Một quá trình đơn luồng chỉ có thể chạy trên một CPU. Đa luồng trên một máy nhiều CPU gia tăng tính đồng hành. Trong kiến trúc đơn xử lý, CPU thường chuyển đổi qua lại giữa mỗi luồng quá nhanh để tạo ra hình ảnh của sự song song nhưng trong thực tế chỉ một luồng đang chạy tại một thời điểm.

IV.3.3 Luồng người dùng và luồng nhân

Chúng ta vừa mới thảo luận là xem xét luồng như một chiều hướng chung. Tuy nhiên, hỗ trợ luồng được cung cấp hoặc ở cấp người dùng, cho các luồng người dùng hoặc ở cấp nhân, cho các luồng nhân.

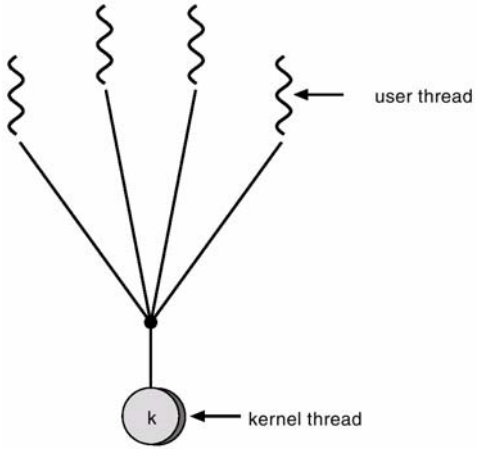
- **Luồng người dùng**: được hỗ trợ dưới nhân và được cài đặt bởi thư viện luồng tại cấp người dùng. Thư viện cung cấp hỗ trợ cho việc tạo luồng, lập thời biểu, và quản lý mà không có sự hỗ trợ từ nhân. Vì nhân không biết các luồng cấp người dùng, tất cả việc tạo luồng và lập thời biểu được thực hiện trong không gian người dùng mà không cần sự can thiệp của nhân. Do đó, các luồng cấp người dùng thường tạo và quản lý nhanh, tuy nhiên chúng cũng có những trở ngại. Thí dụ, nếu nhân là đơn luồng thì bất cứ luồng cấp người dùng thực hiện một lời gọi hệ thống nghẽn sẽ làm cho toàn bộ quá trình bị nghẽn, thậm chí nếu các luồng khác sẵn dùng để chạy trong ứng dụng. Các thư viện luồng người dùng gồm các luồng POSIX Pthreads, Mach C-threads và Solaris 2 UI-threads.
- **Luồng nhân**: được hỗ trợ trực tiếp bởi hệ điều hành. Nhân thực hiện việc tạo luồng, lập thời biểu, và quản lý không gian nhân. Vì quản lý luồng được thực hiện bởi hệ điều hành, luồng nhân thường tạo và quản lý chậm hơn luồng người dùng. Tuy nhiên, vì nhân được quản lý các luồng nếu một luồng thực hiện lời gọi hệ thống nghẽn, nhân có thể lập thời biểu một luồng khác trong ứng dụng thực thi. Trong môi trường đa xử lý, nhân có thể lập thời biểu luồng trên một bộ xử lý khác. Hầu hết các hệ điều hành hiện nay như Windows NT, Windows 2000, Solaris 2, BeOS và Tru64 UNIX (trước Digital UNIX)-hỗ trợ các luồng nhân.

IV.4 Mô hình đa luồng

Nhiều hệ thống cung cấp sự hỗ trợ cả hai luồng nhân và luồng người dùng nên tạo ra nhiều mô hình đa luồng khác nhau. Chúng ta sẽ xem xét ba loại cài đặt luồng thông thường

IV.4.1 Mô hình nhiều-một

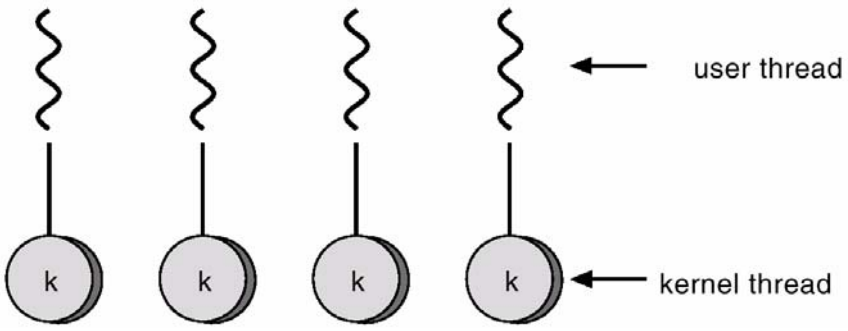
Mô hình nhiều-một (như hình IV.2) ánh xạ nhiều luồng cấp người dùng tới một luồng cấp nhân. Quản lý luồng được thực hiện trong không gian người dùng vì thế nó hiệu quả nhưng toàn bộ quá trình sẽ bị khóa nếu một luồng thực hiện lời gọi hệ thống khóa. Vì chỉ một luồng có thể truy xuất nhân tại một thời điểm nên nhiều luồng không thể chạy song song trên nhiều bộ xử lý. Green threads-một thư viện luồng được cài đặt trên các hệ điều hành không hỗ trợ luồng nhân dùng mô hình nhiều-một.



Hình 0-2-Mô hình nhiều-một

IV.4.2 Mô hình một-một

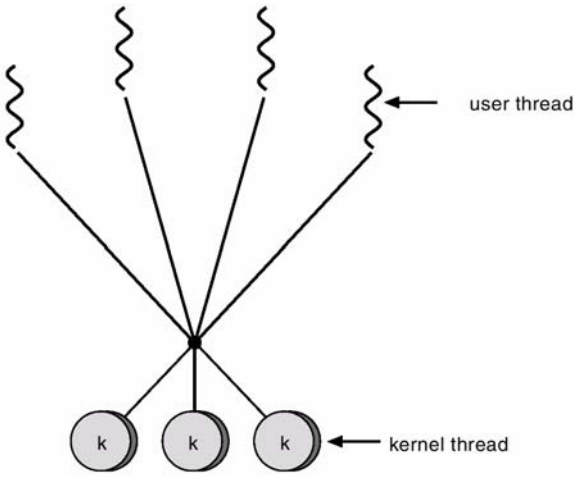
Mô hình một-một (hình IV.3) ánh xạ mỗi luồng người dùng tới một luồng nhân. Nó cung cấp khả năng đồng hành tốt hơn mô hình nhiều-một bằng cách cho một luồng khác chạy khi một luồng thực hiện lời gọi hệ thống nghẽn; nó cũng cho phép nhiều luồng chạy song song trên các bộ xử lý khác nhau. Chỉ có một trở ngại trong mô hình này là tạo luồng người dùng yêu cầu tạo một luồng nhân tương ứng. Vì chi phí cho việc tạo luồng nhân có thể đè nặng lên năng lực thực hiện của ứng dụng, các cài đặt cho mô hình này giới hạn số luồng được hỗ trợ bởi hệ thống. Windows NT, Windows 2000 và OS/2 cài đặt mô hình một-một này.



Hình 0-3-Mô hình một-một

IV.4.3 Mô hình nhiều-nhiều

Mô hình nhiều-nhiều (như hình VI.4) đa hợp nhiều luồng cấp người dùng tới số lượng nhỏ hơn hay bằng các luồng nhân. Số lượng các luồng nhân có thể được xác định hoặc một ứng dụng cụ thể hay một máy cụ thể (một ứng dụng có thể được cấp nhiều luồng nhân trên một bộ đa xử lý hơn trên một bộ đơn xử lý). Trong khi mô hình nhiều-một cho phép người phát triển tạo nhiều luồng người dùng như họ muốn, thì đồng hành thật sự là không đạt được vì nhân có thể lập thời biểu chỉ một luồng tại một thời điểm. Mô hình một-một cho phép đồng hành tốt hơn nhưng người phát triển phải cẩn thận không tạo ra quá nhiều luồng trong một ứng dụng. Mô hình nhiều-nhiều gặp phải một trong hai vấn đề khiếm khuyết: người phát triển có thể tạo nhiều luồng người dùng khi cần thiết và các luồng nhân tương ứng có thể chạy song song trên một bộ đa xử lý. Khi một luồng thực hiện một lời gọi hệ thống khóa, nhân có thể lập thời biểu một luồng khác thực thi. Solaris 2, IRIX, HP-UX, và Tru64 UNIX hỗ trợ mô hình này.



Hình 0-4-Mô hình nhiều-nhiều

IV.5 Cấp phát luồng

Trong phần này chúng ta thảo luận các cấp phát xem xét với các chương trình đa luồng.

IV.5.1 Lời gọi hệ thống fork và exec

Trong chương trước chúng ta mô tả lời gọi hệ thống *fork* được dùng để tạo một quá trình bản sao riêng như thế nào. Trong một chương trình đa luồng, ngữ nghĩa của các lời gọi hệ thống *fork* và *exec* thay đổi. Nếu một luồng trong lời gọi chương trình fork thì quá trình mới sao chép lại quá trình tất cả luồng hay là một quá trình đơn luồng mới? Một số hệ thống UNIX chọn hai ấn bản fork, một sao chép lại tất cả luồng và một sao chép lại chỉ luồng được nạp lên lời gọi hệ thống fork. Lời gọi hệ thống exec điển hình thực hiện công việc trong cùng một cách như được mô tả trong chương trước. Nghĩa là, nếu một luồng nạp lời gọi hệ thống *exec*, chương trình được xác định trong tham số exec sẽ thay thế toàn bộ quá trình-chứa tất cả luồng và các quá trình tải nhẹ.

Việc sử dụng hai ấn bản fork phụ thuộc vào ứng dụng. Nếu exec bị hủy tức thì sau khi phân nhánh (forking) thì sự sao chép lại tất cả luồng là không cần thiết khi chương trình được xác định trong các tham số exec sẽ thay thế quá trình. Trong trường hợp này, việc sao chép lại chỉ gọi luồng hợp lý. Tuy nhiên, nếu quá trình riêng biệt này không gọi exec sau khi phân nhánh thì quá trình riêng biệt này nên sao chép lại tất cả luồng.

IV.5.2 Sự hủy bỏ luồng

Hủy một luồng là một tác vụ kết thúc một luồng trước khi nó hoàn thành.Thí dụ, nếu nhiều luồng đang tìm kiếm đồng thời thông qua một cơ sở dữ liệu và một luồng trả về kết quả, các luồng còn lại có thể bị hủy. Một trường hợp khác có thể xảy ra khi người dùng nhấn một nút trên trình duyệt web để dừng trang web đang được tải. Thường một trang web được tải trong một luồng riêng. Khi người dùng nhấn nút stop, luồng đang nạp trang bị hủy bỏ.

Một luồng bị hủy thường được xem như luồng đích. Sự hủy bỏ một luồng đích có thể xảy ra hai viễn cảnh khác nhau:

- **Hủy bất đồng bộ**: một luồng lập tức kết thúc luồng đích
- **Hủy trì hoãn**: luồng đích có thể kiểm tra định kỳ nếu nó sắp kết thúc, cho phép luồng đích một cơ hội tự kết thúc trong một cách có thứ tự.

Sự khó khăn của việc hủy này xảy ra trong những trường hợp khi tài nguyên được cấp phát tới một luồng bị hủy hay một luồng bị hủy trong khi việc cập nhật dữ liệu xảy ra giữa chừng, nó đang chia sẻ với các luồng khác. Điều này trở nên đặc biệt khó khăn với sự hủy bất đồng bộ. Hệ điều hành thường đòi lại tài nguyên hệ thống từ luồng bị hủy nhưng thường nó sẽ không đòi lại tất cả tài nguyên. Do đó, việc hủy một luồng bất đồng bộ có thể không giải phóng hết tài nguyên hệ thống cần thiết.

Một chọn lựa khác, sự hủy trì hoãn thực hiện bằng một luồng báo hiệu rằng một luồng đích bị hủy. Tuy nhiên, sự hủy sẽ xảy ra chỉ khi luồng đích kiểm tra để xác định nếu nó được hủy hay không. Điều này cho phép một luồng kiểm tra nếu nó sẽ bị hủy tại điểm nó có thể an toàn bị hủy. Pthreads gọi những điểm như thế là các điểm hủy (cancellation points).

Hầu hết hệ điều hành cho phép một quá trình hay một luồng bị hủy bất đồng bộ. Tuy nhiên, Pthread API cung cấp sự hủy trì hoãn. Điều này có nghĩa rằng một hệ điều hành cài đặt Pthread API sẽ cho phép sự hủy có trì hoãn.

IV.5.3 Tín hiệu quản lý

Một **tín hiệu** (signal) được dùng trong hệ điều hành UNIX thông báo một sự kiện xác định xảy ra. Một tín hiệu có thể được nhận hoặc đồng bộ hoặc bất đồng bộ phụ thuộc mã và lý do cho sự kiện đang được báo hiệu. Một tín hiệu hoặc đồng bộ hoặc bất đồng bộ đều theo sau cùng mẫu:

- Tín hiệu được phát sinh bởi sự xảy ra của một sự kiện xác định.
- Tín hiệu được phát sinh được phân phát tới một quá trình.
- Khi được phân phát xong, tín hiệu phải được quản lý.

Một thí dụ của tín hiệu đồng bộ gồm một truy xuất bộ nhớ không hợp lệ hay chia cho 0. Trong trường hợp này, nếu một chương trình đang chạy thực hiện một trong các hoạt động này, một tín hiệu được phát sinh. Các tín hiệu đồng bộ được phân phát tới cùng một quá trình thực hiện thao tác gây ra tín hiệu (do đó lý do chúng được xem đồng bộ).

Khi một tín hiệu được phát sinh bởi một sự kiện bên ngoài tới một quá trình đang chạy, quá trình đó nhận tín hiệu bất đồng bộ. Thí dụ, tín hiệu kết thúc quá trình với phím xác định (như <control><C>) hay thời gian hết hạn. Điển hình, một tín hiệu bất đồng bộ được gởi tới quá trình khác.

Mỗi tín hiệu có thể được quản lý bởi một trong hai bộ quản lý:

- Bộ quản lý tín hiệu mặc định
- Bộ quản lý tín hiệu được định nghĩa bởi người dùng

Mỗi tín hiệu có một bộ quản lý tín hiệu mặc định được thực thi bởi nhân khi quản lý tín hiệu. Hoạt động mặc định có thể được ghi đè bởi một hàm quản lý tín hiệu được định nghĩa bởi người dùng. Trong trường hợp này, hàm được định nghĩa bởi người dùng được gọi để quản lý tín hiệu hơn là hoạt động mặc định. Cả hai tín hiệu đồng bộ và bất đồng bộ có thể được quản lý trong các cách khác nhau. Một số tín hiệu có thể được bỏ qua (như thay đổi kích thước của cửa sổ); các tín hiệu khác có thể được quản lý bằng cách kết thúc chương trình (như truy xuất bộ nhớ không hợp lệ).

Quản lý tín hiệu trong những chương trình đơn luồng không phức tạp; các tín hiệu luôn được phân phát tới một quá trình. Tuy nhiên, phân phát tín hiệu là phức tạp hơn trong những chương trình đa luồng, như một quá trình có nhiều luồng. Một tín hiệu nên được phân phát ở đâu?

Thông thường, các tuỳ chọn sau tồn tại:

- Phân phát tín hiệu tới luồng mà tín hiệu áp dụng
- Phân phát tín hiệu tới mỗi luồng trong quá trình.
- Phân phát tín hiệu tới các luồng cụ thể trong quá trình.
- Gán một luồng xác định để nhận tất cả tín hiệu cho quá trình.

Phương pháp cho việc phân phát tín hiệu phụ thuộc vào loại tín hiệu được phát sinh. Thí dụ, các tín hiệu đồng bộ cần được phân phát tới luồng đã phát sinh ra tín hiệu và không phân phát tới luồng nào khác trong quá trình. Tuy nhiên, trường hợp với tín hiệu bất đồng bộ là không rõ ràng. Một số tín hiệu bất đồng bộ - như tín hiệu kết thúc một quá trình (thí dụ:<control><c>)- nên được gởi tới tất cả luồng. Một số ấn bản đa luồng của UNIX cho phép một luồng xác định tín hiệu nào sẽ được chấp nhận và tín hiệu nào sẽ bị khoá. Do đó, một vài tín hiệu bất đồng bộ có thể được phân phát tới chỉ các luồng không khoá tín hiệu. Tuy nhiên, vì tín hiệu cần được quản lý chỉ một lần, điển hình một tín hiệu được phân phát chỉ luồng đầu tiên được tìm thấy trong một luồng mà không nghẽn tín hiệu. Solaris 2 cài đặt bốn tuỳ chọn; nó tạo một luồng xác định trong mỗi quá trình cho quản lý tín hiệu. Khi một tín hiệu bất đồng bộ được gởi tới một quá trình, nó được gởi tới luồng xác định, sau đó nó phân phát tín hiệu tới luồng đầu tiên không khoá tín hiệu.

Mặc dù Windows 2000 không cung cấp rõ sự hỗ trợ tín hiệu, nhưng chúng có thể được mô phỏng sử dụng lời gọi thủ tục bất đồng bộ (asynchronous produce calls-APC). Tiện ích APC cho phép luồng người dùng xác định hàm được gọi khi luồng người dùng nhận thông báo về một sự kiện xác định. Như được hiển thị bởi tên của nó, một APC rất giống tín hiệu bất đồng bộ trong UNIX. Tuy nhiên, UNIX phải đấu tranh với cách giải quyết tín hiệu trong môi trường đa luồng, phương tiện APC phức tạp hơn như một APC được phân phát tới luồng xác định hơn quá trình.

IV.5.4 Nhóm luồng

Trong phần VI.3, chúng ta mô tả kịch bản đa luồng của một trình phục vụ web. Trong trường hợp này, bất cứ khi nào trình phục vụ nhận một yêu cầu, nó tạo một luồng riêng để phục vụ yêu cầu đó. Ngược lại, tạo một luồng riêng thật sự cao hơn tạo một quá trình riêng, dù sao một trình phục vụ đa luồng có thể phát sinh vấn đề. Quan tâm đầu tiên là lượng thời gian được yêu cầu để tạo luồng trước khi phục vụ yêu cầu, và lượng thời gian xoá luồng khi nó hoàn thành. Vấn đề thứ hai là vấn đề khó giải quyết hơn: nếu chúng ta cho phép tất cả yêu cầu đồng hành được phục vụ trong một luồng mới, chúng ta không thay thế giới hạn trên số lượng luồng hoạt động đồng hành trong hệ thống. Những luồng không giới hạn có thể làm cạn kiệt tài nguyên hệ thống, như thời gian CPU và bộ nhớ. Một giải pháp cho vấn đề này là sử dụng nhóm luồng.

Ý tưởng chung nằm sau nhóm luồng là tạo số lượng luồng tại thời điểm khởi động và đặt chúng vào nhóm, nơi chúng ngồi và chờ công việc. Khi một trình phục vụ nhận một yêu cầu, chúng đánh thức một luồng từ nhóm- nếu một luồng sẵn dùng – truyền nó yêu cầu dịch vụ. Một khi luồng hoàn thành dịch vụ của nó, nó trả về nhóm đang chờ công việc kế. Nếu nhóm không chứa luồng sẵn dùng, trình phục vụ chờ cho tới khi nó rảnh.

Nói cụ thể, các lợi ích của nhóm luồng là:

1) Thường phục vụ yêu cầu nhanh hơn với luồng đã có hơn là chờ để tạo luồng.
2) Một nhóm luồng bị giới hạn số lượng luồng tồn tại bất kỳ thời điểm nào. Điều này đặc biệt quan trọng trên những hệ thống không hỗ trợ số lượng lớn các luồng đồng hành.

Số lượng luồng trong nhóm có thể được đặt theo kinh nghiệm (heuristics) dựa trên các yếu tố như số CPU trong hệ thống, lượng bộ nhớ vật lý và số yêu cầu khách hàng đồng hành. Kiến trúc nhóm luồng tinh vi hơn có thể tự điều chỉnh số lượng luồng trong nhóm dựa theo các mẫu sử dụng. Những kiến trúc như thế cung cấp lợi điểm xa hơn của các nhóm luồng nhỏ hơn-do đó tiêu tốn ít bộ nhớ hơn-khi việc nạp trên hệ thống là chậm.

IV.5.5 Dữ liệu đặc tả luồng

Các luồng thuộc một quá trình chia sẻ dữ liệu của quá trình. Thật vậy, chia sẻ dữ liệu này cung cấp một trong những lợi điểm của lập trình đa luồng. Tuy nhiên, mỗi luồng có thể cần bản sao dữ liệu xác định của chính nó trong một vài trường hợp. Chúng ta sẽ gọi dữ liệu như thế là dữ liệu đặc tả luồng. Thí dụ, trong một hệ thống xử lý giao dịch, chúng ta có thể phục vụ mỗi giao dịch trong một luồng. Ngoài ra, mỗi giao dịch có thể được gán một danh biểu duy nhất. Để gán mỗi luồng với định danh duy nhất của nó chúng ta có thể dùng dữ liệu đặc tả dữ liệu. Hầu hết thư viện luồng gồm Win32 và Pthread – cung cấp một số biểu mẫu hỗ trợ cho dữ liệu đặc tả luồng. Java cũng cung cấp sự hỗ trợ như thế.

IV.6 Pthreads

Pthreads tham chiếu tới chuẩn POSIX (IEEE 1003.1c) định nghĩa API cho việc tạo và đồng bộ luồng. Đây là một đặc tả cho hành vi luồng không là một cài đặt.

Người thiết kế hệ điều hành có thể cài đặt đặc tả trong cách mà họ muốn. Thông thường, các thư viện cài đặt đặc tả Pthread bị giới hạn đối với các hệ thống dựa trên cơ sở của UNIX như Solaris 2. Hệ điều hành Windows thường không hỗ trợ Pthreads mặc dù các ấn bản shareware là sẵn dùng trong phạm vi công cộng.

Trong phần này chúng ta giới thiệu một số Pthread API như một thí dụ cho thư viện luồng cấp người dùng. Chúng ta sẽ xem nó như thư viện cấp người dùng vì không có mối quan hệ khác biệt giữa một luồng được tạo dùng Pthread và luồng được gắn với nhân. Chương trình C hiển thị trong hình dưới đây, mô tả một Pthread API cơ bản để xây dựng một chương trình đa luồng.

Chương trình hiển thị trong hình tạo một luồng riêng xác định tính tổng của một số nguyên không âm. Trong chương trình Pthread, các luồng riêng bắt đầu thực thi trong một hàm xác định. Trong hình, đây là một hàm runner. Khi chương trình này bắt đầu, một luồng riêng điều khiển bắt đầu trong main. Sau khi khởi tạo, main tạo ra luồng thứ hai bắt đầu điều khiển trong hàm runner.

Bây giờ chúng ta sẽ cung cấp tổng quan của chương trình này chi tiết hơn. Tất cả chương trình Pthread phải chứa tập tin tiêu đề pthread.h. pthread_t tid khai báo danh biểu cho luồng sẽ được tạo. Mỗi luồng có một tập các thuộc tính gồm kích thước ngăn xếp và thông tin định thời. Khai báo pthread_attr_t attr hiện diện các thuộc tính cho luồng. Chúng ta sẽ thiết lập các thuộc tính trong gọi hàm pthread_attr_init(&attr). Vì chúng ta không thiết lập rõ thuộc tính, chúng ta sẽ dùng thuộc tính mặc định được cung cấp. Một luồng riêng được tạo với lời gọi hàm pthread_create. Ngoài ra, để truyền định danh của luồng và các thuộc tính cho luồng, chúng ta cũng truyền tên của hàm, nơi một luồng mới sẽ bắt đầu thực thi, trong trường hợp này là hàm runner. Cuối cùng chúng ta sẽ truyền số nguyên được cung cấp tại dòng lệnh, argv[1].

Tại điểm này, chương trình có hai luồng: luồng khởi tạo trong main và luồng thực hiện việc tính tổng trong hàm runner. Sau khi tạo luồng thứ hai, luồng main sẽ chờ cho luồng runner hoàn thành bằng cách gọi hàm pthread_join. Luồng runner sẽ hoàn thành khi nó gọi hàm pthread_exit.

```
#include<pthread>
#include<stdio.h>
int sum: /*Dữ liệu này được chia sẻ bởi thread(s)*/
void *runner(void *param); /*luồng*/

main(int argc, char *argv[])
{
        pthread_t tid; /*định danh của luồng*/
        pthread_attr_t attr; /*tập hợp các thuộc tính*/
        if(argc !=2){
        fprintf(stderr, "usage: a.out <integer value>");
        exit();
}
if (atoi(argv[1] < 0)){
        fprintf(stderr,"%d must be >= 0 \n", atoi(argv[1]));
        exit();
}
/*lấy các thuộc tính mặc định*/
pthread_attr_init(&attr);
/*tạo một luồng*/
pthread_create(&tid,&attr,runner, argv[1]);
```

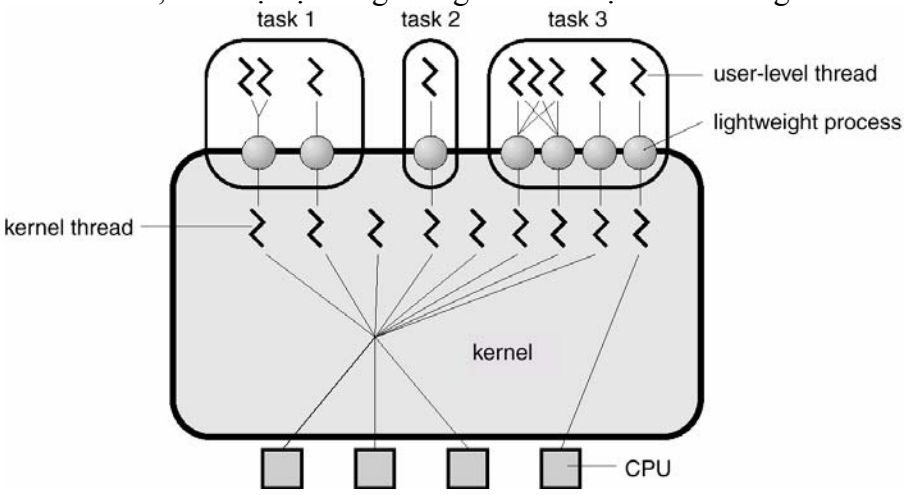
```
/*bây giờ chờ luồng kết thúc*/
pthread_join(tid,NULL);
printf("sum = %d\n",sum);
/*Luồng sẽ bắt đầu điều khiển trong hàm này*/
void *runner(void *param)
{
        int upper = atoi(param);
        int i;
        sum = 0;
        if (upper > 0){
        sum+= i;
}
pthread_exit(0);
}
}
```

Hình 0-5-Chương trình C đa luồng dùng Pthread API

IV.7 Luồng Solaris 2

Solaris 2 là một ấn bản của UNIX với hỗ trợ luồng tại cấp độ nhân và cấp độ người dùng, đa xử lý đối xứng (SMP) và định thời thời thực. Solaris 2 cài đặt Pthread API hỗ trợ luồng cấp người dùng với thư viện chứa APIs cho việc tạo và quản lý luồng (được gọi luồng UI). Sự khác nhau giữa hai thư viện này rất lớn, mặc dù hầu hết người phát triển hiện nay chọn thư viện Pthread. Solaris 2 cũng định nghĩa một cấp độ luồng trung gian. Giữa luồng cấp nhân và cấp người dùng là các quá trình nhẹ (lightweight process- LWPs). Mỗi quá trình chứa ít nhất một LWP. Thư viện luồng đa hợp luồng người dùng trên nhóm LWP cho quá trình và chỉ luồng cấp người dùng hiện được nối kết tới một LWP hoàn thành công việc. Các luồng còn lại bị khoá hoặc chờ cho một LWP mà chúng có thể thực thi trên nó.

Luồng cấp nhân chuẩn thực thi tất cả thao tác trong nhân. Mỗi LWP có một luồng cấp nhân, và một số luồng cấp nhân (kernel) chạy trên một phần của nhân và không có LWP kèm theo (thí dụ, một luồng phục vụ yêu cầu đĩa). Các luồng cấp nhân chỉ là những đối tượng được định thời trong hệ thống. Solaris 2 cài mô hình nhiều-nhiều; toàn bộ hệ thống luồng của nó được mô tả trong hình dưới đây:



Hình 0-6-Luồng Solaris 2

Các luồng cấp người dùng có thể giới hạn hay không giới hạn. Một luồng cấp người dùng giới hạn được gán vĩnh viễn tới một LWP. Chỉ luồng đó chạy trên LWP và yêu cầu LWP có thể được tận hiến tới một bộ xử lý đơn (xem luồng trái nhất trong hình trên). Liên kết một luồng có ích trong trường hợp yêu cầu thời gian đáp ứng nhanh, như ứng dụng thời thực. Một luồng không giới hạn gán vĩnh viễn tới bất kỳ LWP nào. Tất cả các luồng không giới hạn được đa hợp trong một nhóm cac LWP sẵn dùng cho ứng dụng. Các luồng không giới hạn là mặc định. Solaris 8 cũng cung cấp một thư viện luồng thay đổi mà mặc định chúng liên kết tới tất cả các luồng với một LWP.

Xem xét hệ thống trong hoạt động: bất cứ một quá trình nào có thể có nhiều luồng người dùng. Các luồng cấp người dùng này có thể được định thời và chuyển đổi giữa LWPs bởi thư viện luồng không có sự can thiệp của nhân. Các luồng cấp người dùng cực kỳ hiệu quả vì không có sự hỗ trợ nhân được yêu cầu cho việc tạo hay huỷ, hay thư viện luồng chuyển ngữ cảnh từ luồng người dùng này sang luồng khác.

Mỗi LWP được nối kết tới chính xác một luồng cấp nhân, ngược lại mỗi luồng cấp người dùng là độc lập với nhân. Nhiều LWPs có thể ở trong một quá trình, nhưng chúng được yêu cầu chỉ khi luồng cần giao tiếp với một nhân. Thí dụ, một LWP được yêu cầu mỗi luồng có thể khoá đồng hành trong lời gọi hệ thống. Xem xét năm tập tin khác nhau-đọc các yêu cầu xảy ra cùng một lúc. Sau đó, năm LWPs được yêu cầu vì chúng đang chờ hoàn thành nhập/xuất trong nhân. Nếu một tác vụ chỉ có bốn LWPs thì yêu cầu thứ năm sẽ không phải chờ một trong những LWPs để trả về từ nhân. Bổ sung một LWP thứ sáu sẽ không đạt được gì nếu chỉ có đủ công việc cho năm.

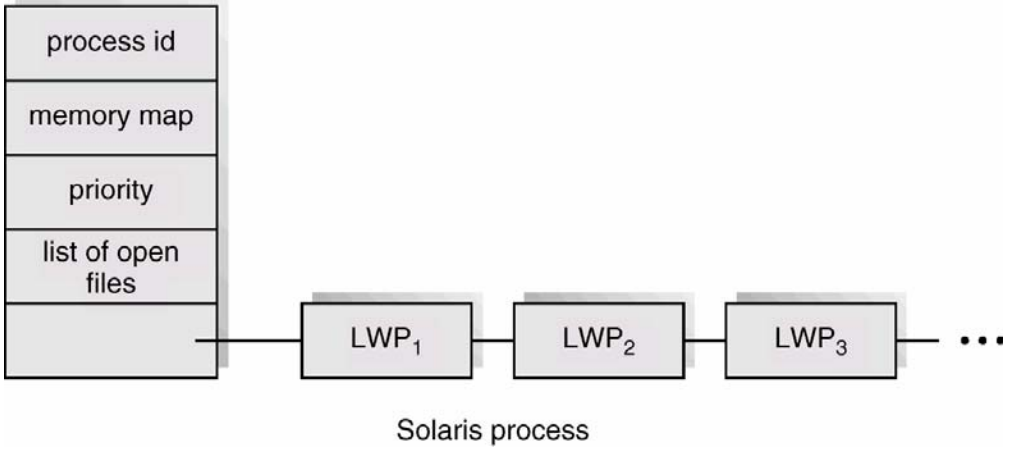
Các luồng nhân được định thời bởi bộ lập thời biểu của nhân và thực thi trên một hay nhiều CPU trong hệ thống. Nếu một luồng nhân khoá (trong khi chờ một thao tác nhập/xuất hoàn thành), thì bộ xử lý rảnh để thực thi luồng nhân khác. Nếu một luồng bị khoá đang chạy trên một phần của LWP thì LWP cũng khoá. Ở trên vòng, luồng cấp người dùng hiện được gán tới LWP cũng bị khoá. Nếu một quá trình có nhiều hơn một LWP thì nhân có thể định thời một LWP khác.

Thư viện luồng tự động thay đổi số lượng LWPs trong nhóm để đảm bảo năng lực thực hiện tốt nhất cho ứng dụng. Thí dụ, nếu tất cả LWPs trong một quá trình bị khoá bởi những luồng có thể chạy thì thư viện tự tạo một LWP khác được gán tới một luồng đang chờ. Do đó, một chương trình được ngăn chặn từ một chương trình khác bởi sự nghèo nàn của những LWPs không bị khoá. LWPs là những tài nguyên nhân đắt để duy trì nếu chúng không được dùng. Thư viện luồng “ages” LWPs và xoá chúng khi chúng không được dùng cho khoảng thời gian dài, điển hình khoảng 5 phút. Các nhà phát triển dùng những cấu trúc dữ liệu cài đặt luồng trên Solaris 2:

- Luồng cấp người dùng chứa một luồng ID; tập thanh ghi (gồm một bộ đếm chương trình và con trỏ ngăn xếp); ngăn xếp; và độ ưu tiên (được dùng bởi thư viện cho mục đích định thời). Không có cấu trúc dữ liệu nào là tài nguyên nhân; tất cả chúng tồn tại trong không gian người dùng.
- Một LWP có một tập thanh ghi cho luồng cấp nhân nó đang chạy cũng như bộ nhớ và thông tin tính toán. Một LWP là một cấu trúc dữ liệu nhân và nó nằm trong không gian nhân
- Một luồng nhân chỉ có một cấu trúc dữ liệu nhân và ngăn xếp. Cấu trúc dữ liệu gồm bản sao các thanh ghi nhân, con trỏ tới LWP mà nó được gán, độ ưu tiên và thông tin định thời.

Mỗi quá trình trong Solaris 2 gồm nhiều thông tin được mô tả trong khối điều khiển quá trình (Process Control Block-PCB). Trong thực tế, một quá trình Solaris 2

chứa một định danh quá trình (Process ID-PID); bản đồ bộ nhớ; danh sách các tập tin đang mở, độ ưu tiên; và con trỏ của các luồng nhân vơi quá trình (Hình).



Hình 0-7-Quá trình Solaris 2

IV.8 Luồng Windows 2000

Windows 2000 cài đặt Win32 API. Win32 API là một API chủ yếu cho họ hệ điều hành Windows (Windows 95/98/NT và Windows 2000). Thực vậy, những gì được đề cập trong phần phần này áp dụng tới họ hệ điều hành này

Ứng dụng Windows chạy như một quá trình riêng rẻ nơi mỗi quá trình có thể chứa một hay nhiều luồng. Windows 2000 dùng ánh xạ một-một nơi mà mỗi luồng cấp người dùng ánh xạ tới luồng nhân được liên kết tới.Tuy nhiên, Windows cũng cung cấp sự hỗ trợ cho một thư viện có cấu trúc (fiber library) cung cấp chức năng của mô hình nhiều-nhiều. Mỗi luồng thuộc về một quá trình có thể truy xuất một không gian địa chỉ ảo của quá trình.

Những thành phần thông thường của một luồng gồm:

- ID của luồng định danh duy nhất luồng
- Tập thanh ghi biểu diễn trạng thái của bộ xử lý
- Ngăn xếp người dùng khi luồng đang chạy ở chế độ người dùng. Tương tự, mỗi luồng cũng có một ngăn xếp nhân được dùng khi luồng đang chạy trong chế độ nhân
- Một vùng lưu trữ riêng được dùng bởi nhiều thư viện thời gian thực và thự viện liên kết động (DLLs).

Tập thanh ghi, ngăn xếp và vùng lưu trữ riêng được xem như ngữ cảnh của luồng và được đặc tả kiến trúc tới phần cứng mà hệ điều hành chạy trên đó. Cấu trúc dữ liệu chủ yếu của luồng gồm:

- RTHREAD (executive thread block-khối luồng thực thi).
- KTHREAD (kernel thread-khối luồng nhân)
- TEB (thread environment block-khối môi trường luồng)

Các thành phần chủ yếu của RTHREAD gồm một con trỏ chỉ tới quá trình nào luồng thuộc về và địa chỉ của thủ tục mà luồng bắt đầu điều khiển trong đó. ETHREAD cũng chứa một con trỏ chỉ tới KTHREAD tương ứng.

KTHREAD gồm thông tin định thời và đồng bộ hóa cho luồng. Ngoài ra, KTHREAD chứa ngăn xếp nhân (được dùng khi luồng đang chạy trong chế độ nhân) và con trỏ chỉ tới TEB.

ETHREAD và KTHREAD tồn tại hoàn toàn ở không gian nhân; điều này có nghĩa chỉ nhân có thể truy xuất chúng. TEB là cấu trúc dữ liệu trong không gian người dùng được truy xuất khi luồng đang chạy ở chế độ người dùng. Giữa những trường khác nhau, TEB chứa ngăn xếp người dùng và một mảng cho dữ liệu đặc tả luồng (mà Windows gọi là lưu trữ cục bộ luồng)

IV.9 Luồng Linux

Nhân Linux được giới thiệu trong ấn bản 2.2. Linux cung cấp một lời gọi hệ thống *fork* với chức năng truyền thống là tạo bản sao một quá trình. Linux cũng cung cấp lời gọi hệ thống *clone* mà nó tương tự như tạo một luồng. c*lone* có hành vi rất giống như *fork*, ngoại trừ thay vì tạo một bản sao của quá trình gọi, nó tạo một quá trình riêng chia sẻ không gian địa chỉ của quá trình gọi. Nó chấm dứt việc chia sẻ không gian địa chỉ của quá trình cha mà một tác vụ được nhân bản đối xử giống rất nhiều một luồng riêng rẻ.

Chia sẻ không gian địa chỉ được cho phép vì việc biểu diễn của một quá trình trong nhân Linux. Một cấu trúc dữ liệu nhân duy nhất tồn tại cho mỗi quá trình trong hệ thống. Một cấu trúc dữ liệu nhân duy nhất tồn tại cho mỗi quá trình trong hệ thống. Tuy nhiên, tốt hơn lưu trữ dữ liệu cho mỗi quá trình trong cấu trúc dữ liệu là nó chứa các con trỏ chỉ tới các cấu trúc dữ liệu khác nơi dữ liệu này được được lưu. Thí dụ, cấu trúc dữ liệu trên quá trình chứa các con trỏ chỉ tới các cấu trúc dữ liệu khác hiện diện danh sách tập tin đang mở, thông tin quản lý tín hiệu, và bộ nhớ ảo. Khi fork được gọi, một quá trình mới được tạo cùng với một bản sao của tất cả cấu trúc dữ liệu của quá trình cha được liên kết tới. Khi lời gọi hệ thống *clone* được thực hiện, một quá trình mới chỉ tới cấu trúc dữ liệu của quá trình cha, do đó cho phép quá trình con chia sẻ bộ nhớ và tài nguyên của quá trình cha. Một tập hợp cờ được truyền như một tham số tới lời gọi hệ thống *clone*. Tập hợp cờ này được dùng để hiển thị bao nhiêu quá trình cha được chia sẻ với quá trình con. Nếu không có cờ nào được đặt, không có chia sẻ xảy ra và clone hoạt động giống như fork. Nếu tất cả năm cờ được đặt, quá trình con chia sẻ mọi thứ với quá trình cha. Sự kết hợp khác của cờ cho phép các cấp độ chia sẻ khác nhau giữa hai mức độ cao nhất này.

Điều thú vị là Linux không phân biệt giữa quá trình và luồng. Thật vậy, Linux thường sử dụng thuật ngữ tác vụ-hơn là quá trình hay luồng-khi tham chiếu tới dòng điều khiển trong chương trình. Ngoài quá trình được nhân bản, Linux không hỗ trợ đa luồng, cấu trúc dữ liệu riêng hay thủ tục nhân. Tuy nhiên, những cài đặt Pthreads là sẵn dùng cho đa luồng cấp người dùng.

IV.10 Luồng Java

Như chúng ta đã thấy, hỗ trợ cho luồng có thể được cung cấp tại cấp người dùng với một thư viện như Pthread. Hơn nữa, hầu hết hệ điều hành cung cấp sự hỗ trợ cho luồng tại cấp nhân. Java là một trong số nhỏ ngôn ngữ cung cấp sự hỗ trợ tại cấp ngôn ngữ cho việc tạo và quản lý luồng. Tuy nhiên, vì các luồng được quản lý bởi máy ảo Java, không bởi một thư viện cấp người dùng hay nhân, rất khó để phân cấp luồng Java như cấp độ người dùng hay cấp độ nhân. Trong phần này chúng ta trình bày các luồng Java như một thay đổi đối với mô hình người dùng nghiêm ngặt hay mô hình

cấp nhân. Phần sau chúng ta sẽ thảo luận một luồng Java có thể được ánh xạ tới luồng nhân bên dưới như thế nào.

Tất cả chương trình tạo ít nhất một luồng điều khiển đơn. Thậm chí một chương trình Java chứa chỉ một phương thức main chạy như một luồng đơn trong máy ảo Java. Ngoài ra, Java cung cấp các lệnh cho phép người phát triển tạo và thao tác các luồng điều khiển bổ sung trong chương trình.

IV.10.1 Tạo luồng

Một cách để tạo một luồng rõ ràng là tạo một lớp mới được phát sinh từ lớp Thread và viết đè phương thức run của lớp Thread. Tiếp cận này được hiển thị trong hình sau, ấn bản Java của chương trình đa luồng xác định tổng các số nguyên không âm.

Một đối tượng của lớp phát sinh sẽ chạy như một luồng điều khiển đơn trong máy ảo Java. Tuy nhiên, tạo một đối tượng được phát sinh từ lớp Thread không tạo một luồng mới, trái lại phương thức start mới thật sự tạo luồng mới. Gọi phương thức start cho đối tượng mới thực hiện hai thứ:

1) Nó cấp phát bộ nhớ và khởi tạo một luồng mới trong máy ảo Java.
2) Nó gọi phương thức run, thực hiện luồng thích hợp để được chạy bởi máy ảo Java. (Chú ý, không thể gọi phương thức run trực tiếp, gọi phương thức start sẽ gọi phương thức run)

```
Class Summation extends Thread
{
        public Summation (int n){
                upper   = n;
        }
        public void run(){
                int sum = 0;

                if (upper>0){
                        for(int i = 1; i<= upper; i++){
                                sum+=i;
                }
                System.out.println("The sum of "+upper+ " is " + sum);
        }
        private int upper;
}

public class ThreadTester
{
        public static void main(String args[]){
                if(args.length>0){
                        if(Integer.parseInt(args[0])<0)
                                System.err.println(args[0] + " must be >= 0.");
                        else{
                                Summation thrd = new Summation
(Integer.parseInt(args[0]));
                                Thrd.start();
                        }
```

```
                }
                Else
                        System.out.println("Usage: summation < integer value");
        }
}
```

Hình 0-8- Chương trình Java để tính tổng số nguyên không âm

Khi chương trình tính tổng thực thi, hai luồng được tạo bởi JVM. Luồng đầu tiên là luồng được nối kết với ứng dụng-luồng này bắt đầu thực thi tại phương thức main. Luồng thứ hai là luồng Summation được tạo rõ ràng với phương thức start. Luồng Summation bắt đầu thực thi trong phương thức run của nó. Luồng kết thúc khi nó thoát khỏi phương thức run của nó.

IV.10.2 JVM và hệ điều hành chủ

Cài đặt điển hình của JVM ở trên đỉnh của hệ điều hành chủ (host operating system). Thiết lập này cho phép JVM che giấu chi tiết cài đặt của hệ điều hành bên dưới và cung cấp môi trường không đổi, trừu tượng cho phép chương trình Java hoạt động trên bất kỳ phần cứng nào hỗ trợ JVM. Đặc tả cho JVM không hiển thị các luồng Java được ánh xạ tới hệ điều hành bên dưới như thế nào để thay thế việc quên đi việc quyết định cài đặt cụ thể của JVM. Windows 95/98/NT và Windows 2000 dùng mô hình một-một; do đó, mỗi luồng Java cho một JVM chạy trên các hệ điều hành này ánh xạ tới một luồng nhân. Solaris 2 khởi đầu cài đặt JVM dùng mô hình nhiều-một. Tuy nhiên, ấn bản 1.1 của JVM với Solaris 2.6 được cài đặt dùng mô hình nhiều-nhiều.

IV.11 Tóm tắt

Luồng là một dòng điều khiển trong phạm vi một quá trình. Quá trình đa luồng gồm nhiều dòng điều khiển khác nhau trong cùng không gian địa chỉ. Những lợi điểm của đa luồng gồm đáp ứng nhanh đối với người dùng, chia sẻ tài nguyên trong quá trình, tính kinh tế, và khả năng thuận lợi trong kiến trúc đa xử lý.

Luồng cấp người dùng là các luồng được nhìn thấy bởi người lập trình và không được biết bởi nhân. Thư viện luồng trong không gian người dùng điển hình quản lý luồng cấp người dùng. Nhân của hệ điều hành hỗ trợ và quản lý các luồng cấp nhân. Thông thường, luồng cấp người dùng nhanh hơn luồng cấp nhân trong việc tạo và quản lý. Có ba loại mô hình khác nhau liên quan đến luồng cấp người dùng và luồng cấp nhân. Mô hình nhiều-một ánh xạ nhiều luồng người dùng tới một luồng nhân. Mô hình một-một ánh xạ mỗi luồng người dùng tới một luồng nhân tương ứng. Mô hình nhiều-nhiều đa hợp nhiều luồng người dùng tới một số lượng nhỏ hơn hay bằng luồng nhân.

Những chương trình đa luồng giới thiệu nhiều thử thách cho việc lập trình, gồm ngữ nghĩa của lời gọi hệ thống fork và exec. Những vấn đề khác gồm huỷ bỏ luồng, quản lý tín hiệu, và dữ liệu đặc tả luồng. Nhiều hệ điều hành hiện đại cung cấp nhân hỗ trợ luồng như Windows NT, Windows 2000, Solaris 2 và Linux. Pthread API cung cấp tập hợp các hàm để tạo và quản lý luồng tại cấp người dùng. Java cung cấp một API tương tự cho việc hỗ trợ luồng. Tuy nhiên, vì các luồng Java được quản lý bởi JVM và không phải thư viện luồng cấp người dùng hay nhân, chúng không rơi vào loại luồng người dùng hay nhân.